PSis - Programação de Sistemas - 2015/2016
Exame 1ª Época, 6 de Junho de 2016, 11h00, Duração: 2h

Sistemas operativos

1. Uma das caraterísticas oferecidas pelos sistemas operativos é o isolamento.

1.a) [1 valor] Indique dois casos em que se observa isolamento na operação normal do sistema operativo.

Isolamento entre processos: a memória e recurso utilizados pelos processos encontra-se compartimentado e necessitam de autorização para ser partilhados
Isolamento entre processo e recursos físicos: o acesso aos recursos não é feito directamente, mas sim através da mediação do kernel.

1.b) [1 valor] Das arquiteturas de sistemas operativos existentes, indique aquela que é usada actualmemente na maioria dos Sistemas Operativos e que permite a implementação dos diversos tipos de isolamento. Justifique.

Arquitectura em camadas.
A organização do Sistema em camadas permite a implemenatçao de isolament descrito anteriormente. As aplicações encontram-se numa camada superior e todos os acessos a recursos (que podem ser partilhados ou devem ser guardados) são feitos e geridos através de uma camada inforior, que verifica todas as permissões.

Processos

2. [1 valor] Quais os mecanismos do UNIX/LINUX disponibilizados aos programadores para que um processo possa ser notificado e possa processar o encerramento de um processo filho? Justifique.

Sinais: quando um filho ternina o seu pai é notificado através de um sinal o programador por registar um *handler* para esse sinal.
Função wait. o programador pode invocar a função wait que bloqueia até que um dos filhos termine

3. [1 valor] Aquando da terminação de um processo como podem ser comunicados dados entre esse processo e o seu parente? Justifique e exemplifique.

Quando um processo termina, por exemplo através da chamada sistema **exit**, o processo que termina pode comunicar um valor inteiro que será entregue ao processo pai.
O processo pai espera pela terminação do filho invocando a função **wait**. Esta função "retorna" o valor que o filho enviou através do exit.

Memória partilhada

4. [1.5 valores] Explique de que modo o suporte de memória virtual fornecido pelos processadores é fundamental na implementação de memória partilhada entre processos.

Como o espaço de endereçamento de cada processo é diferente, é complicado arranjar um endereço (correspondente à memória partilhada) que fosse compatível com todos os processos.
A memória virtual permite o uso de diversos endereços diferente (potencialmente em processos diferentes) que apontem para a mesma página/segmento de memória.
os mecanismos de memória partilhada também permitem o controlo de acesso às páginas partilhadas

5. [1 valor] Descreva os passos que ocorrem quando um processo tenta aceder a um endereço de memória "inválido".

Sempre que um processo acede a uma posiçõade memória (por exemplo mov AX, M[Bx]) é necessário fazer uma tradução entre endereço virtual (presente no exemplo em BX) e o endereços freal.
Essa tradução é realizada numa tabela do processador. tabela que é actualizada pelo kernel com os limites da memória acessível por um processo.
Quando o CPU verifica que o endereços a ser acedido não está compatível com a informação nessa tabela gera uma interrupção.
Esse interrupção é tratada pelo kernel, que a encamina para o processso. O processo recebe um sinal.

IPC

6. Os FIFOs e o sockets Stream/UNIX permitem a comunicação entre dois processos localizados numa mesma máquina.

6.a) [1 valor] Descreva duas semelhanças na programação e/ou utilização deste mecanismos de IPC

O nome ou endereços são nomes/caminhos de ficheiros
Ambas podem usar a funcões read/write para ler e escrever.
Ambas implementam o conceito de stream

6.b) [1 valor] Descreva duas diferenças na programação e/ou utilização deste mecanismos de IPC

No caso dos sockets é necessário fazer connect, no caso do FIFOs basta umm open
No caso do FIFOs, qualquer processo pode ler ou escrever. No caso dos sockets a comunicação é unicamente entre o processo que fez o accept e o que fez o connect (e filhos).
Os sockets são bi-direcionais. os FIFOs são unidirecionais

Sincronização

7. Considere um programa concorrente composto por duas *threads* que partilham uma variável inteira (`k_shared`), usada para transferir um valor entre threads· Todas as outras variáveis são locais a cada uma das *threads*.

```
...
while (1){
    aux_1 = gera_valor();
    k_shared = aux_1;
}
```

```
...
while(1){
    aux_2 = k_shared;
    if(aux_2 != old_aux){
        old_aux = aux_2;
        processa_valor(aux_2);
    }
}
```

7.a) [2 valores] Considerando unicamente as linhas `k_shared = aux_1` e `aux_2 = k_shared`, indique se é necessário a guarda destas instruções usando um *mutex* imediatamente antes e depois das mesmas. Justifique a sua resposta.

O mutex à volta destas duas instruções não serve para nada, visto a escrita e leitura de inteiros ser atómica e não poderem ser executadas ao mesmo tempo.

7.b) [1 valor] Qual o problema da espera activa que é utilizada na solução anterior? Justifique.

o problema da espera activa é que ocupa recurso (nomeadamente CPU). Enquanto não chega ummnovo valor o processo está a executar as seguintes instruções:

```
while(1)
aux_2 = k_shared
if(aux2 != old_aux
```

7.c) [2 valores] Altere o código anterior, usando as primitivas de sincronização necessárias, de modo a que seja eliminada a espera activa e que todos os valores gerados na função `gera_valor` sejam processados por `processa_valor.`

```
s_1 = semaphore(0)
s_2 = semaphore(1)
```

```
...
while (1){
    aux_1 = gera_valor();
    s_2.wait();
    k_shared = aux_1;
    s_1.signal()
}
```

```
...
while(1){
    s_1.wait()
    aux_2 = k_shared;
    s_2.signal();
    processa_valor(aux_2);
}
```

Princípios de Desenvolvimento de Software

8. No desenvolvimento de sistemas deve-se seguir um conjunto de princípios. Dois desses princípios são *Separation of concerns* e *Build to change instead of building to last*.

8.a) [1 valor] Descreva ambos os princípios.

Separation of concernes – uma funcionalidade de um sistema deve estar implementada unicamente num componente, não estando distribuída por diversas partes do sistema.
Build to change – a organização do sistema e divisão de responsabilidades deve facilitar alterações futuras: correcões de bugs, otimizações, extensões ao sistema

8.b) [1 valor] Explique de que modo a sua implementação correcta de *separation of concerns* facilita o a prossecução do princípio *Build to change*.

Ao garantir que uma determinada funcionalidade está limitada a um componete, e que existe um interface bem definida, alterações nessa funcionalidade serão implementadas de forma mais fácil e levarão a menos efeitos colaterais.

Arquiteturas de Software

9. [1.5 valores] Os padrões (ou estilos) arquiteturais podem ser divididos em três grandes classes. Indique as três classes, e para cada uma descreva-a e dê um exemplo de padrão dessa classe

Estruture
- como desenhar os componentes, como estruturar o código
- Exemplo: layered, OO, data-abstraction, State transition system

Deployment
- como distribuir os componentes pelos vários nós de computação
- Peer-to-peer, client-server, n-tier

Comunicação
- que mecanismo de alto nível de comnicação de irá usar
- RPC, Message queus, pipes, eventos

Testes de software

10. [2 valores] Pretende-se implementar uma função que verifica se uma string contêm um endereço de email válido (`int verifica_email(char * str)`). A função retorna 1 se a string contiver um email válido e zero caso contrário.
Defina, apresente e descreva o conjunto de testes unitários que permitem a verificação do código dessa função.

Verificação se retorna 0 nos seguintes casos (os erros são detectados):
- Verificar se função dá erro quando string é vazia (ouponteiro a NULL)
- Vereificar se @ está em falra ou se aparece várias vezes
- Verificar se aparecem caracteres inválidos
- Verificar se estrutura do mail é incorrecta (domínio) deposi do @

Verificação se retorna 1 quando endereço está correcto

11. [1 valor] Que tipo de erros são encontrados quando se executa um teste de regressão (*Regression test*)?

Erros que são introduzidos om alterações ao código já realizado:
- introdução de novas funcionalidades
- correcções de erros
- ...